ESM客户端Linux版本

代码地址：

老版本：26.XX.XX <http://193.168.10.222/svn/esm/linux/UnixRav>

新版本：3.0.x.x [http://193.168.10.222/svn/esm/linux/UnixRav](http://193.168.10.222/svn/esm/linux/UnixRav2017)3

**老版本代码：26.xx.xx**

**代码结构：**

.

├── buildrav.sh 编译脚本

├── Makefile

├── readme.txt

├── unixpubsoft 第三方库的位置

│   ├── ~~libengcpp~~  一个编译libcpp.so的库，已废弃

│   ├── lib\_tools\_64 存放编译好的 libtoolscfg.so, libtoolsunit.so

│   ├── pubtool 第三方的开源库

│   └── x11 X11的依赖库

└── wxrav

├── Makefile

├── Makefile.conf 编译依赖的配置

├── Makefile.template 配置的模板，根据此模板动态生成Makefile.conf

├── packetgui 打包的一些脚本

├── ravnet 后台的服务，包括网络版和ESM

├── ravunix 启动界面的GUI主程序

├── setup 界面的GUI安装程序

├── uninstall 卸载脚本

├── unixengapi 工具常用的一些类，主要为引擎提供的

├── unixpub 常用的库，主要为序列号验证，安装卸载库，工具库

└── wrd 命令行界面的程序

**编译前准备：**

1 首先确认要编译的版本类型， 老版本支持若干种版包，包括 ：

L27I glibc>=2.12 32位

L25I glibc>=2.5 32位

LINTELX64 glibc>=2.12 64位

LMIPSX64 glibc>=2.12 64位

AARCH glibc>=2.12 64位

LMIPS glibc>=2.12 32位

以上依赖是官方最终发布所依赖的最小环境，并不是在普通机器上编译的版本。

在编译的时候，可以打开 buildrav.sh 确定要编译的版本，及配置相应的gcc

打开 buildrav.sh , 第58行，从此行开始可以配置相应的编译项，找到相关版本下，

platform32=1 编译32位版本， 0/空 编译64

TOOLCHAIN: gcc的前缀，工具链编译用， 比如：i686-centos6-linux-gnu-

HOST：编译的host选项，跨平台或工具链用， 比如：arm-linux

2 gcc编译选项，在 UnixRav/wxrav/Makefile.template 里， 第二行

EXEFLAG=-static-libstdc++ -static-libgcc 此句是指定静态链接gcc和stdc++,如果不确定本机是否有相关静态库，则屏掉它

EXEFLAG=#-static-libstdc++ -static-libgcc

3 文件准备

在uninstall目录，有6个文件依赖于 ravnet/esmagent 目录，为了避免文件不同步，所以在此建立软链接

**编译步骤：**

1 普通用户下，创建文件夹，如UnixRavWorkDir，此文件夹做为编译目录，后面将用WORKDIR 来表示此目录

2 从SVN同步代码

svn co <http://193.168.10.222/svn/esm/linux/UnixRav> --username=xxx –password=xxxx

3 将病毒组提供的引擎，拷贝至路径

├── unix\_engine2013

│   ├── rame

│   └── rame\_LINTELX64\_64.bin (病毒组提供的应该是rame.bin, 修改为相应版 本,LINTELX64为编译的版本，64为CPU位数)

4 建立 virusdb 目录，并将病毒库拷贝至此

├── malware.rmd 病毒库文件

└── vlstd-linux.cfg 病毒库版本配置

5 完成后，目录结构为：

├── unix\_engine2013

│   ├── rame

│   └── rame\_LINTELX64\_64.bin

├── UnixRav

│   ├── buildrav.sh

│   ├── Makefile

│   ├── readme.txt

│   ├── unixpubsoft

│   └── wxrav

└── virusdb

├── malware.rmd

└── vlstd-linux.cfg

3 进入UnixRav目录，打开 buildrav.sh, 确认相应版本的编译选项正常

4 运行 ./buildrav.sh 26.03.50 LINTELX64 NOL $(WORKDIR)

此过程就开始编译，首先编译x11的库，然后编译 pubtools里的第三方库，如果出错，则查看相应三方库的出错信息，解决不能编译成功的问题，如果顺利进行，则会进入到代码编译，但这个大概率是编译不成功的，需要做几处修改

如果第三方库在编译的时候出现了问题，可以查看出错记录，进入具体的第三方为中单独编译，比如如果在wxX11-2.6.3中出现了问题，那么进入到此目录

查看目录下是否mk.sh文件

没有， 运行 cp ../pathch/mksh/wxX11-2.6.3.sh ./mk.sh

运行 */bin/sh ./mk.sh gcc g++ ‘’*

如果出错了，查明原因，修改后继续编译，直至 $(WORKDIR)/ravlocal/lib 下生成了 libwx.so

5 完成后，目录结构

├── bin

├── cdrom

├── cdrom.26.03.50.LINTELX64.NOL.tgz 完成后的安装包

├── lib

├── ravlib\_LINTELX64

├── ravlocal -> /home/rising/pangxl//ravlib\_LINTELX64

├── result

├── tools

├── unix\_engine2013

├── UnixRav

├── update.ini

├── upload ESM升级文件的位置

└── virusdb

**新版本 3.0 编译：**

新版本代码结构：

.

├── Authorization 序列号检证模块

├── BaseController 界面和后台服务基础类

├── CmakeLists.txt cmake配置文件

├── CommonAPI 常用功能API

├── Communicator 和中心的通讯模块

├── config.h 配置文件

├── ContextMenu 右键菜单模块

├── DefenMod 文件监控的通讯模块

├── Kernel 内核模块

├── Netfilter 网络监控的通讯模块

├── PackageTools 打包的工具

├── RavConsole 字符界面

├── RavDaemon 后台服务

├── RavQt 界面程序

├── ScanEngine 引擎的封装

├── Setup 安装及卸载模块

├── Storage 存储模块

├── ThirdPart 第三方模块，包括zlib, libiconv, libxml2, openssl, libcurl, ncurses

├── unix\_engine2013 杀毒引擎， 由病毒组提供

├── Upgrade 更新模块

**编译准备：**

编译依赖于cmake, qt5, 所以需要下载安装包或者编译相关代码等。

**编译步骤：**

1 安装cmake

cmake 版本必须大于3.1, 直接编译安装至系统目录即可

2 安装 qt

qt 版本必须为qt5, 建议使用qt-everywhere-opensource-src-5.5.0, 因为这个是使用g++98编译的，利于跨平台编译

QT的编译脚本（建议）：

./configure -opensource -prefix /home/rising/qt/build -release -no-c++11 -shared -no-largefile -no-pkg-config -qt-zlib -qt-libpng -no-eglfs -no-directfb -no-linuxfb -qt-xcb -no-libjpeg -no-freetype -no-openssl -no-nis -no-cups -no-glib -no-dbus -no-xcursor -no-xfixes -no-xrandr -no-xrender -no-separate-debug-info -nomake examples -nomake tools -nomake tests -no-iconv -no-sql-sqlite -no-sql-mysql -no-qml-debug -no-gif -confirm-license -no-opengl -no-harfbuzz -no-dbus -no-compile-examples -no-evdev -skip qtactiveqt -skip qtconnectivity -skip qtdeclarative -skip qtdoc -skip qtenginio -skip qtgraphicaleffects -skip qtimageformats -skip qtlocation -skip qtmultimedia -skip qtquick1 -skip qtquickcontrols -skip qtscript -skip qtsensors -skip qtserialport -skip qtsvg -skip qttools -skip qttranslations -skip qtwebkit -skip qtwebkit-examples -skip qtwebsockets -skip qtxmlpatterns

这个编译脚本去除了相应的无关选项

3 创建工作目录 WORKDIR

4 从SVN同步代码

svn co <http://193.168.10.222/svn/esm/linux/UnixRav>3 --username=xxx --password=xxxx

5 编译相关的第三方库

3.0版本的依赖库和26版本比较相似，编译方式也相似，故在此不做重新编译可以将以前编译的拷贝至此

├── libcrypto.a （openssl)

├── libcurl.a (libcurl)

├── libQt5Core.so (QT)

├── libQt5Gui.so (QT)

├── libQt5Widgets.so (QT)

├── libQt5XcbQpa.so (QT)

├── libqxcb.so (QT)

├── libravcrypto.so (openssl)

├── libravcurl.so (libcurl)

├── libraviconv.so (libiconv)

├── libravncurses.so (ncurses)

├── libravssl.so (openssl)

├── libravxml2.a (libxml2)

├── libravxml2.so (libxml2)

├── libravz.a (libz)

├── libravz.so (libz)

└── libssl.a (openssl)

这些库，可以自己编译，也可以从250平台下载已编译好的

250平台地址：

<http://193.168.10.222/svn/esm/linux/unixrav2017/build/thirdpart>

将相应平台的库拷贝至 $(WORKDIR)/UnixRav3/ThirdPart/lib

6 拷贝引擎代码

将引擎 unix\_engine2013 拷贝至 $(WORKDIR)/UnixRav3 下

250平台地址：

[http://193.168.10.222/svn/esm/linux/unixrav2017/u](http://193.168.10.222/svn/esm/linux/unixrav2017/build/thirdpart)nix\_engine2013

7 创建编译目录，在$(WORKDIR)下创建 build目录

8 修改编译配置文件$(WORKDIR)/UnixRav3/CmakeLists.txt

set(QT5\_LIBRARY\_PATH "XX") 将此值设置为自己的QT编译目录

set(BUILD\_OUTPUT\_DIRECTORY “XX”) 将此值设置为自己的编译目录

set(DEBUG 0)

如果不确定本机有GCC静态库，把这句屏掉

set(CMAKE\_EXE\_LINKER\_FLAGS "-static-libgcc -static-libstdc++")

9 进入编译目录$(WORKDIR)/build，运行

cmake ../UnixRav3

make

编译完成后

./3.0.0.1/

├── bin

│   ├── autorepair.bin

│   ├── autoupdate.bin

│   ├── downloader.bin

│   ├── rav.bin

│   ├── RAVGJCZ0

│   ├── RAVGLTX0

│   ├── RAVGLTX2

│   ├── RAVRJJH0

│   ├── RAVRJJH00

│   ├── RAVRJJH2

│   ├── RAVRJJH4

│   ├── RAVRJJH6

│   ├── ravsetup.bin

│   ├── ravuninst.bin

│   └── RAVZDFY0

├── lib

│   ├── libravauth.a

│   ├── libravcommon.so

│   ├── libravcommunicator.so

│   ├── libravdefenbinder.so

│   ├── libravfilterbinder.so

│   ├── libravqtsupport.so

│   ├── libravscanengine.so

│   ├── libravstorage.so

│   └── libravupgrade.so

└── tools

├── crossupdater.bin

└── PackTools

这些就是编译生成的文件，然后打包生成相应的安装包

打包没有做成 shell脚本，而是做成了python脚本布署在服务器上，提供WEB页面管理

打包平台:

[http://193.168.10.5](http://193.168.10.5/) rising/rising123

后台ssh密码， rising/Rising!@#$1234

**支持平台：**

老版本：aarch64, mips64, x86\_64, i686, AIX5/6/7, solaris

新版本： aarch64, mips64, x86\_64

运行环境：

kernel > 2.6 , glibc>=2.12